2016-06

Controlling robotic swarm behavior utilizing real-time kinematics and artificial physics

Armandt, David

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CONTROLLING ROBOTIC SWARM BEHAVIOR UTILIZING REAL-TIME KINEMATICS AND ARTIFICIAL PHYSICS

by

David Armandt

June 2016

Thesis Advisor:	Richard Harkins
Second Reader:	Brian Bingham

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2016	Master's Thesis 12-01-2014 to 06-01-2016

4. TITLE AND SUBTITLE

CONTROLLING ROBOTIC SWARM BEHAVIOR UTILIZING REAL-TIME KINEMATICS AND ARTIFICIAL PHYSICS

5. FUNDING NUMBERS

6. AUTHOR(S)

David Armandt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Recent commercial developments in small, low cost real-time kinematic GPS sensors enable position realization for light, mobile platforms with centimeter-level accuracy. With a high degree of positional confidence, we explore the feasibility of close-proximity operations of cooperative autonomous agents in an outdoor, GPS-enabled environment. A computer-simulated hookian spring force is used to control a "swarm" of robotic agents, a technique called artificial physics. The computer model applies a proportional spring constant based off position information, and the resultant force is applied to each agent respectively. We validate the model by comparing it to analytic solutions, then further refine the model by comparing it to field testing data. With an accurate model of the system, user-defined tasks are tested in simulations and the same algorithm then controls the behavior of the robotic swarm in an outdoor environment.

14. SUBJECT TERMS

Multiple robot coordination, robotic swarm, robotic swarm behavior, real-time Kinematic GPS, Robot Operating System

15. NUMBER OF PAGES 71

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

CONTROLLING ROBOTIC SWARM BEHAVIOR UTILIZING REAL-TIME KINEMATICS AND ARTIFICIAL PHYSICS

David Armandt
Lieutenant, United States Navy
B.S., Rensselaer Polytechnic Institute, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2016

Approved by: Richard Harkins
Thesis Advisor

Brian Bingham
Second Reader

Kevin Smith
Chair, Department of Physics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Recent commercial developments in small, low cost real-time kinematic GPS sensors enable position realization for light, mobile platforms with centimeter-level accuracy. With a high degree of positional confidence, we explore the feasibility of close-proximity operations of cooperative autonomous agents in an outdoor, GPS-enabled environment. A computer-simulated hookian spring force is used to control a "swarm" of robotic agents, a technique called artificial physics. The computer model applies a proportional spring constant based off position information, and the resultant force is applied to each agent respectively. We validate the model by comparing it to analytic solutions, then further refine the model by comparing it to field testing data. With an accurate model of the system, user-defined tasks are tested in simulations and the same algorithm then controls the behavior of the robotic swarm in an outdoor environment.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

DoD	Department of Defense
NPS	Naval Postgraduate School
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vessel
USN	U.S. Navy
GPS	Global Positioning Satellite
ROS	Robotic Operating System
RTK	Real Time Kinematics
IMU	Inertial Measurement Unit
COTS	Commercial-off-the-Shelf
MEMs	Micro Electro-Mechanical Systems

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

Previous research at NPS explored the use of physics-based computer algorithms to control the behavior of a multi-agent unmanned aerial vehicle (UAV) system in a laboratory setting. This paper explores a variation of that technique that can be used in outdoor, GPS-enabled environments. Analytic predictions inform computer simulations, which support live field experimentation. Data from the real-world experimentation is then analyzed and the computer simulation is refined to provide a more accurate model of the actual system.

The intent of this research is developing a multiple autonomous agent system in an outdoor, GPS environment where the agents can operate in close-proximity to each other. To do this, a computer-simulated virtual spring connects each of the agents in the system. By adjusting the "unstretched" length of the spring, the individual agents move until all the virtual spring forces experienced by that agent are in equilibrium. The result is a system that self-organizes into a formation centered around a mobile base station.

This type of self-organizing formation behavior can be thought of as a swarm. Robotic swarm behavior relieves the burden of an operator directly controlling the actions of each individual agent, while still accomplishing a desired outcome to be performed by the system. This allows the operator to focus on high-level tasking and lets the swarm behavior algorithm determine how best to accomplish that tasking.

To accomplish a model of this behavior, a computer algorithm calculates the distance between one agent and every other agent in the system separately, and applies a proportional spring constant to the difference. This creates an artificial "spring force" between one agent and all the other agents in the system separately. The corresponding forces are added as a vector sum, and a total resultant force is designated to the corresponding agent. The force is converted into an acceleration, the acceleration is integrated into a velocity, and the velocity is integrated into a position. This is repeated for every agent in the system. The agents in the simulation move to their new position for each time step of the loop, and the new positions of each of the agents are used for the next iteration. The process is repeated until the net force on each agent is zero, the agents stop moving, and the system is at equilibrium.

Performing this in an outdoor, GPS environment, the same computer-simulated spring force

algorithm is used. However, real-world position and orientation information of the individual agents are given as inputs to the model. This research utilizes real-time Kinematics (RTK) GPS to obtain accurate position information and onboard inertial measurement units (IMUs) for orientation information of the agent. After calculating the net force and the subsequent desired velocity of each individual agent, a velocity command is sent to each respective agent, and its updated RTK GPS position is used in the next iteration of the spring force algorithm.

Acknowledgments

First and foremost, I would like to thank my wife for her constant support and encouragement. I would also like to thank my advisor, Richard Harkins, for making it possible for this project to happen and my second reader/co-advisor, Dr. Brian Bingham, for his mentorship and guidance during critical periods of the process.

A huge amount of gratitude also goes out to Aurelio Monarrez, Steve Jacobs and Matt Clement. Without your help, this project would never have been able to get off the ground. Finally, I thank God for His unceasing grace and for every opportunity given to me to make this world a better place.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Motivation for Swarm Research

The Department of Defense (DoD) has identified military applications of autonomous systems as a major force multiplier. Extensive efforts in the fielding of these systems make them a viable option for front-line commanders. From 2005 to 2011, there was a 1,200% increase in Unmanned Aerial Vehicle (UAV) combat air patrols and more flight hours conducted by UAVs than by manned strike aircraft [1]. The funding has followed, although at a slower pace, with an increase in DOD spending on unmanned system from $284M in FY2000 to $5.6B in FY2013 [2].

Research and Development in the area of autonomous systems is also enthusiastically gaining momentum. The Office of Naval Research recently demonstrated the capability of unmanned, self-guided boats to escort a vessel and swarm a target [3]. The Defense Advance Research Project Agency launched its Anti-submarine Warfare Continuous Trail Unmanned Vessel demonstrator in January 2016 [4]. Unmanned systems are gaining popularity not only in the DOD, but in commercial applications as well. Amazon began testing delivery orders via multi-rotor UAVs in April 2015 [5] and Google had logged over 1.5 million miles by self-driven cars as of April 2016 [6].

The present-day commonality of autonomous systems leads to a pressing problem for the Navy. How will unmanned systems affect the way future wars are fought at sea? The Naval strategy for the past 75 years remains unchanged, centering on power projection at sea with an aircraft carrier. The carrier battlegroup commander has significant advantage over the commander of a conventional surface ship because embarked aircraft offer range superiority over the enemy. However, as technology advances, leaders must focus their efforts on what will give the greatest maritime advantage in the future.

Unmanned Systems can offer a number of advantages over traditional manned systems, such as decreased risk to personnel, long dwell time, no life support systems required, and,

with a satellite data link and pre-staged assets, can offer an immediate physical presence anywhere in the world without a constantly deployed fleet. The computer and technology industry advances have made owning and operating unmanned systems much more accessible than they were even five years ago. However, inexpensive unmanned systems must be coordinated en masse to “swarm” and overwhelm an enemy’s defenses in order to have the same effect as larger, more expensive and complex weapons.

Although the low material cost and easy access to online open source content lowers the barrier for entry into the field of robotics, the integration of various platforms and sensors to operate autonomously is still a challenging problem. The primary sensor involved in this research is real-time kinematics GPS, where calculation of a GPS-carrier signal differential produces a centimeter-level accurate position vector between modules [7].

In addition to reliable and accurate sensors, effective and efficient control of unmanned system is essential. Control algorithms to accomplish this purpose are often implemented to relieve the requirement of a human operator to directly control the agent(s). For well defined and predictable tasks, predetermined actions are set by the controlling algorithm and are executed by the system, called a “brute force” style of control. For instance, a welding robot arm on a car assembly line exhibits this style of control. The welding spots do not change from car to car, so a programmer ensures the robot makes the same movement each time a new car comes through the assembly line.

However, this “brute force" approach for robotic systems in ill defined and uncontrolled environments is a daunting task; predicting the multitude of potential situations that a system may experience is nearly impossible. The potential number of situations also increases as more agents are added to the system. A different approach is developed by looking at natural physical phenomena.

Natural laws of physics govern interactions between particles both big and small in the universe. Complex structures arise out of particles obeying simple physical forces, such as galaxies forming due to gravity and DNA molecules arising from interactions of the electromagnetic force. By programming robotic agents to react to simulated forces, similar

types of behavior can be realized.

1.2 Previous Work

Directing physical agents toward a location with the lowest computer-simulated potential is a concept called potential field method (PFM) first proposed by Hogan [8] to navigate a manipulator arm around obstacles in a fixed environment. Problems with this approach are explored by Koren and Borenstein [9]. One major problem is that while the individual agents are moving toward the lowest potential in a system, an agent could get caught in a local potential minimum that may not be the lowest possible potential of the system. If the agent does not have enough energy to get out of the local minimum, the agent will never be able to reach the global minimum, and the desired behavior of the system is never realized. One possible solution to this problem is explored in Section 2.3.

One of the frameworks utilizing an PFM approach has been developed for distributed control of swarms of robots by Spears et al. [10] called physicomimetics. Using natural physical laws as a guide, physicomimetics is a method whereby a computer imposed interaction between agents form the basis for the movement of each individual agent. Instead of pre-determining the entire potential field of the given environment, physicomimetics introduces algorithm based virtual forces that are imposed on the agents in a system.

This approach is also called artificial physics (AP) because although the interactions and applied forces are all virtual, the agents behave just as particles under the influence of physical laws in the real-world. Apker and Potter in [11] used the laws governing the states of matter, solids, liquids and gases, to have the robotic agents execute different types of behavior for their system.

The key issue is choosing the sensor information that the computer algorithm will use to determine the swarm behavior. Ma'sum et al. [12] used three quadrotor aircraft in a similar network architecture to the one presented in this study. However, they used computer vision for their main tracking sensor. Difficulties in using computer vision and LIDAR-based

sensor information to determine robot localization are presented in Chapter 3.

Numerous research groups use a high speed, near-infrared camera system, called Vicon motion capture [13], to track the motion of reflective markers on an object with millimeter accuracy. The result is a 3D computer generated object with position and orientation information in the Vicon coverage area. Kushleyev et al. at the GRASP Lab at University of Pennsylvania [14] utilize this approach (with some fantastic video at http://youtube.com/user/TheDmel/videos). While amazing, it requires significant amount of installed infrastructure and cannot easily be replicated in an outdoor environment.

Another very common position realization technique is GPS, and Alvissalim et al. [15] place a GPS receiver on multiple quadrotors and use the GPS coordinates to self-deploy and extend the coverage of a Wi-Fi network. The approach is similar, but since the individual quadrotors are nominally separated by large distances to affect an extended Wi-Fi network, each quadrotor's location does not need to be extremely accurate since the chance of collision is small.

Given the potential benefits that robotics can offer to the maritime domain, this thesis explores the use of various sensors and platforms in an artificial physics environment to perform a task requiring a high level of precision. To demonstrate the capability of the system, the scenario will be having multiple UAVs take off and follow in close-proximity to an Unmanned Surface Vessel (USV).

CHAPTER 2:
Artificial Physics Approach

The approach used in this study is a system of agents (agents) experiencing attractive and repulsive forces to maintain a certain desired distance from nearby agents. The end state of the system will be one where each agent rests in a potential well, which will be determined by the combination of attractive and repulsive forces felt by a single agent. This simple concept will allow for systematic formation of a collection of mobile robots.

The artificial physics environment is set up so that the separate agents experience a "force" of attraction or repulsion depending on the requirement of the task. This "force" is imposed by a computer algorithm, which causes a physical reaction from the platform. For example, if the operator wanted a UAV to fly toward an object, (s)he would set up the algorithm so that the UAV "felt" an attractive force in the direction of the object.

2.1 A Physics Foundation

The basis for the desired actions of the individual agents is firmly rooted in well-defined classical physics. Summoning Newton's second law of motion, the force exerted on a particle will accelerate the particle ($\vec{F} = m\vec{a}$). The agents start in random locations in a specified 2D grid. For our purposes, we will treat each agent as a particle point mass with $m = 1$ kg.

The previous research at NPS [16] utilized a gravity-like force expressed as

$$\vec{F} = G\frac{m_1 m_2}{r^2}\hat{r} \tag{2.1}$$

where m_1 and m_2 are the masses of the two agents and r is the distance between them. In the previous research, the G constant was a user-controlled variable that could be adjusted through a keypad on a handheld device. Since gravity is an attractive-only force, the algorithm had a "switch over" distance, where the force would change from being attractive to being repulsive to prevent the agents from colliding with each other. This $\frac{1}{r}$ potential is not ideal, since the "force" felt by the agent increases as the agent gets closer to the

desired final location at the bottom of the potential well and decreases the farther away it is. In practice, that means the agent will have sharper and more drastic movements when it is near a neighboring agent's location, whereas the agent will have very faint movements when it is very far away.

A better solution for such a system could be modeled as a mass-spring system. In this case, an agent that is very far from its optimum spacing will have a very strong force pushing or pulling it, and when the agent is near its final desired location, the force will be much smaller. The artificial force on an agent from another agent then becomes a simple application of Hooke's law

$$\vec{F}_{i,j} = -k(\vec{x}_{i,j} - \vec{x}_0) \tag{2.2}$$

where k is the spring constant that controls how strong the force of attraction or repulsion acts, $x_{i,j}$ is the distance between the ith agent and the jth agent, and x_0 is an optimum spacing between the agents (Figure 2.1).

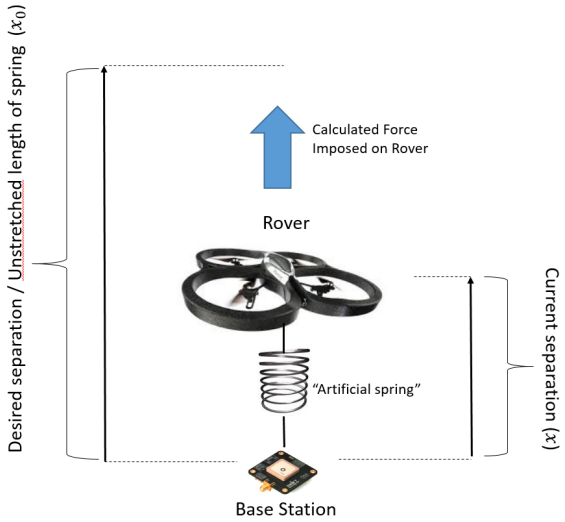

Figure 2.1: Diagram of Artificial Spring Force on an Agent.

Consider only the $\hat{x}$ components in Cartesian coordinates of a system of agents. By summing up the $\hat{x}$ forces acting on a single agent from all other agents, we get the total net force in the $\hat{x}$ direction that is acting on a particular agent

$$\vec{F}_{i,net} = \sum_{j=1}^{N} -k(x_{i,j} - x_o)\hat{x} \tag{2.3}$$

where N is the total number of agents in the system.

We are modeling this as linear system and the principle of superposition holds. That allows us to predict that the overall force on the agent from two or more forces is the sum of the result that would have been caused by each force individually. In addition, once the force is known, we can analyze of dynamics of a single agent as if it were a single mass-spring system by itself, and the total dynamics of the system as a whole is the aggregate sum. The outcome of the computer simulation will let us know if our model accurately portrays the analytic solution.

The oscillatory nature of the spring force invites us to introduce viscous damping term in order to reach a stable equilibrium more expeditiously

$$\vec{F}_{i,total} = \vec{F}_{i,net} - \beta \frac{dx_i}{dt}\hat{x} \tag{2.4}$$

where β is the damping coefficient.

The $\vec{F}_{total}$ is calculated for every agent in the swarm, taking care not to calculate a spring force exerted by the agent on itself!

2.2 Numerical Integration

Once the total force in the $\hat{x}$ direction on a agent is known, we can then determine the $\hat{x}$ behavior of that agent based off of classical mechanics.

A common technique used to calculate trajectories of particles in molecular dynamics computer simulations is the Verlet Velocity algorithm. This numerical method for integrating differential equations calculates the velocity and position of the particle at the same value of the time variable.

$$\vec{a}(t+dt) = \frac{\vec{F}_{total}}{m} \tag{2.5}$$

$$\vec{v}(t+dt) = \vec{v}(t) + \frac{1}{2}(\vec{a}(t) + \vec{a}(t+dt))dt \tag{2.6}$$

$$\vec{x}(t+dt) = \vec{x}(t) + \vec{v}(t)dt + \frac{1}{2}\vec{a}(t)dt^2 \tag{2.7}$$

The same procedure is done for the $\hat{y}$ component in Cartesian coordinates, giving a 2D solution for the system for given values of k and β.

2.3 Analytic Prediction

To determine values for k and β that give reasonable solutions for the scenario, we begin by looking for an analytic solution. Since this is a multi-body problem, an exact analytic solution (especially one with arbitrary number of agents with arbitrary mass) can be difficult, if not impossible, to obtain. We start by making the approximation that the system as a whole can be described by a single agent exhibiting damped harmonic motion, and compare those results to the results from a computer simulation.

Assuming the net force on an single agent is a simple spring, Equation 2.4 now becomes

$$\vec{F}_{total} = \vec{F}_{sping} - \vec{F}_{damping} = -k(x - x_o)\hat{x} - \beta\frac{d\vec{x}}{dt} = m\vec{a}\,. \tag{2.8}$$

Or, in the form of a differential equation,

$$m\frac{d^2x}{dt^2} + \beta\frac{dx}{dt} + kx = 0\,, \tag{2.9}$$

this homogeneous second-order differential equation has solutions of the form

$$x = e^{\lambda t} \,. \tag{2.10}$$

The auxiliary equation for λ is then

$$m\lambda^2 + \beta\lambda + k = 0 \,, \tag{2.11}$$

and the roots for the quadratic equation are

$$\lambda = \frac{-\beta \pm \sqrt{\beta^2 - 4mk}}{2m} \,. \tag{2.12}$$

The discriminant tells the type of system that we would expect to see. If it is positive, the system is overdamped. If negative, the system is underdamped. If the discriminant is 0, then the system is critically damped.

The values for k and β are completely user-defined, since there is no "actual" spring, it is simply a virtual force. For example, a spring constant $k = 2$ N/m (and $m = 1$ kg) would mean for a critically damped system

$$\beta^2 - 4(1)(2) = 0 \tag{2.13}$$

β would have to be 2.82 kg/s to make the system critically damped. A smaller value would make the system underdamped, and a higher value would make the system overdamped. The same procedure can be done to find k if given a specified β.

2.4 Simulation Results

The MATLAB simulation incorporated an arbitrary number of ten agents, each with a mass = 1 kg. This is easily scalable to include more or heavier agents, but to investigate the effects of the spring constant and the damping coefficient, the number of agents was kept constant throughout.

The agents are started in random locations on a 100x100 meter grid (Figure 2.2).

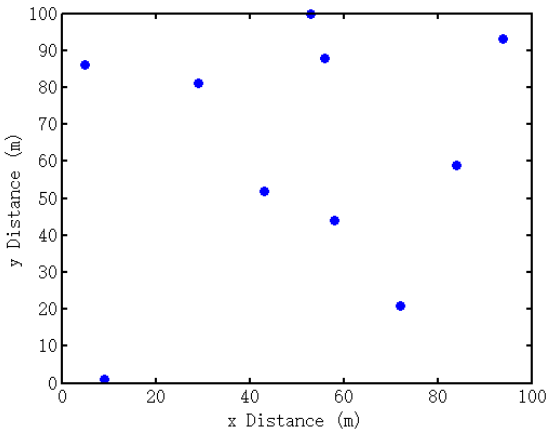

Figure 2.2: Plot Agents at Time=0 Seconds.

From this position, each agent exerts an attractive force on each of the other agents as discussed before.

The attractive force draws the agents closer together (Figure 2.3). The speed with which the agents move is dependent on two factors, the spring constant, k, and the damping coefficient β. We will discuss these more in depth later.

A consequence of Hooke's law in this application is the agents will push apart from each other if they are closer than the optimal spacing distance x_o. Each of the agents is now experiencing both an attractive force and a repulsive force. This causes the agents to self-arrange into a formation in which the spacing between the agents is equal for all agents (Figure 2.4).

Two factors, the spring constant k and the damping coefficient β, influence the agents' ability to arrange into an optimum formation. Since both of these factors are at work in the simulation, our strategy was to hold one factor constant while varying the other, and then switch. To that end, holding the spring constant at a value of $k = 2$N/m, we vary the damping coefficient β (Figure 2.5).

As shown in Figure 2.5, a value of $\beta = 1$ kg/s gives an underdamped solution, where

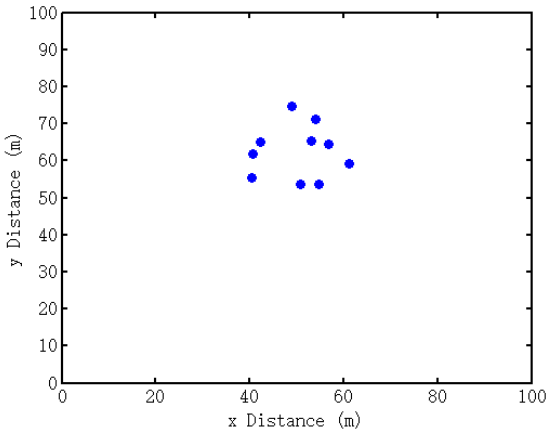

Figure 2.3: Plot Agents at Time=0.5 Seconds.

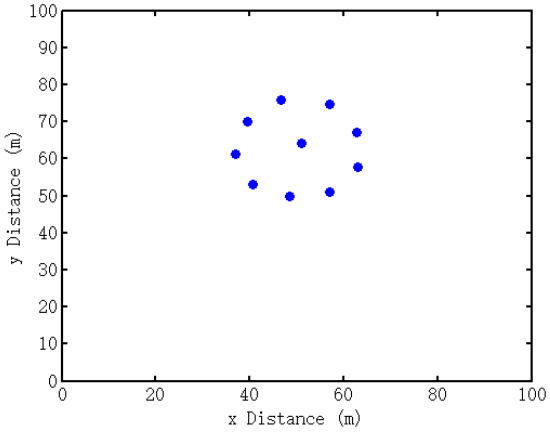

Figure 2.4: Plot Agents at Time=1 Second.

the agents oscillate around the optimized distance and do not reach equilibrium (at least not within a reasonable amount of time!). A value of $\beta = 10$ kg/s gives an overdamped solution, where the agents take a longer time to reach the optimum spacing. The value of $\beta = 4$ kg/s gives a slightly less than critically damped solution, which allows us to see a slight oscillation to the agent, which gives confirmation that the agent is actually at optimum spacing.

Next, the damping coefficient β is held constant and the effect of varying the spring constant k is analyzed (Figure 2.6).

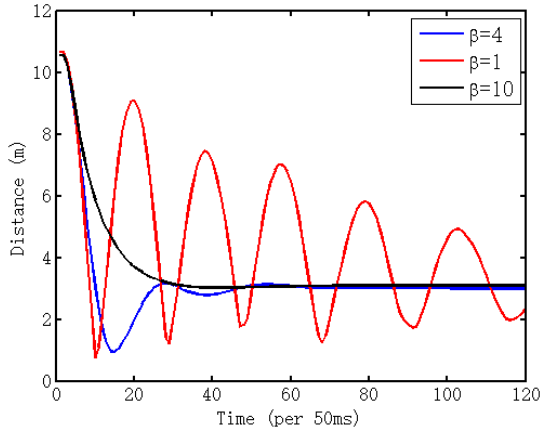

Figure 2.5: Plot of Average agent Distance against Time with Spring Constant k=2 N/m.

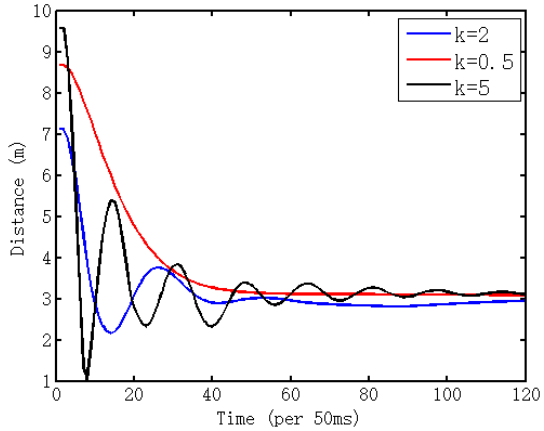

Figure 2.6: Plot of Average agent Distance against Time with Damping Coeff. β=4 kg/s.

The spring constant k is a measure of how strong the force of attraction or repulsion is between any two agents. Again, we can see and underdamped solution (k = 5N/m), an overdamped solution (k = 0.5N/m) and a slightly less than critically damped solution (k = 2N/m).

It is observed that in order for the system to attain equilibrium with the desired spacing between each of the agents, the system has to have a high enough k to overcome viscous

damping, but low enough to settle out in a desired amount of time. The system must also have a high enough damping coefficient β so that the system does not oscillate continuously, but low enough that the agents can actually move.

For the objectives of this simulation, values of $m = 1\text{kg}$, $k = 2\text{N/m}$ and $\beta = 2.8\text{kg/s}$ prove sufficient in organizing agents into an acceptable formation in a reasonable amount of time.

This corresponds almost exactly with our analytic prediction, proving that damped harmonic motion solution is a valid assumption for the computer model.

2.4.1 Problem with PFM

As stated previously, while the agents are moving according to the forces acting on them, a agent could get caught in a potential well that may not be the lowest possible potential of the system. Without sufficient energy to escape, the agent will settle and reach equilibrium in the local minimum potential well. The physical result is that the agents in the system will not be all equally spaced, and the intent of a symmetric formation will never be realized.

With randomly placed agents, this is a very distinct possibility. To simplify the problem, the initial conditions are imposed on the system so the probability for local minima is very low. The initial conditions are such that the agents start symmetrical to each other with the base station in the middle. This ensures that the minima reached for each of the agents at equilibrium is the global minimum for the system.

For systems where the initial conditions cannot be set, the methods outlined in this paper can still be utilized. Many simulations can be run in order to find which simulation has the lowest possible potential, and then compare that with the system in question. If the system in question is not at the lowest possible potential, then one of the agents is most likely "stuck" in a local minimum potential well. One could then perturb the system in an effort to get the agent out of the well.

2.5 Rotational Compensation

The algorithm gives the desired x-y movements that each agent makes in the fixed reference frame of the 2-D plot. However, real-world agents (especially quadrotors) can rotate independently of the fixed reference frame. To account for this difference in heading direction, a rotational transformation matrix is required. If an agent is at a certain location, the artificial physics algorithm determines that the agent needs to move in a certain x-y direction. Given the current angle of the agent's heading relative to a predetermined initial heading (i.e., North), the rotation transformation matrix will convert the required direction into the agent's reference frame, so that the agent's left-right/forward-backward motors can then produce the overall desired action.

The 2-D rotation matrix is as follows:

$$R(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}.$$

Given a desired x and y velocity, v_x and

$$\begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

where θ is the angle of the agent heading with respect to true north, and v'_x and v'_y are the actual velocity commands that are sent to the agent to accomplish the desired behavior in the fixed reference frame.

CHAPTER 3:
Sensor Selection and Hardware Implementation

3.1 Requirements and Available Systems

To replicate the laboratory experimentation in an outdoor environment, a sensor system is required to replicate the position and orientation information that was previously provided by the installed Vicon system. The sensor system must be small and light enough to allow the agent to take off and maneuver, yet accurate enough for the applications such as landing on a small USV platform. We will first discuss the issue of position realization in an uncontrolled, infrastructure-free environment.

There are a few ways to resolve location information in real-world environments. A common tactic is to use a camera with computer vision software. Knowing what the camera expects to see, a programmer can set up an algorithm to determine the agent's location based on visual cues. This is, of course, dependent on line of sight, lighting conditions, visual angle considerations, as well as computing power, since computer vision algorithms can be quite processor intensive depending on the requirements.

Another favored technique used in location determination is wave reflection as seen in SONAR/RADAR/LIDAR. Using a laser, microwave or ultrasonic pulse an object's distance can be determined by the time it takes the pulse to travel to and be reflected from the object. Again, this technique is highly dependent on line of sight, and distance/free-space loss can be a limiting factor. In addition, these sensors allow the identification that an object exists, and not who or what the object is. Prior knowledge of what an operator expects to see is required to identify objects and determine location. The equipment and processing power can also be significant and can be impractical fit onto a small agent.

The most common technique in use today for outdoor manned and unmanned systems is GPS. The only line of sight required is toward the sky, and can be used day or night and in a variety weather conditions. Receivers collect unique signals sent from at least four satellites. The receiver calculates its distance to each of the satellites by determining the

time it took the signal to go from the satellite to the receiver, and ultimately triangulate its position. However, each signal sent from the satellites is about 300 meters in length, and the precision of measuring the signal can vary. Atmospheric effects can also impact the measurement of the signal. The ionosphere slows down the signal from the satellite, but this delay can also vary by time and location. The imprecise signal measurements and also ionospheric disturbances limit the typical accuracy range for standard units to 3–5 meter accuracy.

While this may be good enough for something like driving directions, the desire for close-proximity swarm behavior requires a higher degree accuracy. In the scenario put forth in this paper, the agents are initially close together (about 0.5 meters) on the USV, so required GPS error bars would have to be at least +/- 0.25 meters.

An alternative method for providing more accurate GPS is a technique called real-time kinematics (RTK). It does not provide absolute position, but will provide a highly accurate relative position between two RTK GPS modules.

In addition to measuring the signal from a GPS satellite, an RTK GPS receiver also measures the phase of the carrier wave that the signal is being modulated on. This is called carrier phase tracking. The carrier wavelength is about 19 centimeters, so the measurement of the carrier phase can be much more accurate. However, since the receiver is measuring only the phase and ignoring the content of the transmitted signal, it becomes difficult for the receiver to align the signals or be off by an integer number of wavelengths. This is called integer ambiguity and is usually addressed with statistical methods to reduce the accuracy error.

In addition to the increased accuracy due to carrier phase tracking, and assuming the two modules are close enough so that they are receiving the same amount of ionospheric interference, comparing the carrier phase from one module with the carrier phase received by another module can cancel out the adverse effects of the ionosphere on the signal. Sending the phase data between the two modules is commonly done by a radio modem in the UHF band. Utilizing the differences in the signal received by two GPS units, the relative position accuracy between the two units can be resolved to 1-5 centimeters.

Also, accurate orientation information is required to effectively control the agent. A number of options are available to the user for this as well. A simple digital compass would give the heading of the agent based off of the Earth's magnetic field. Inertial Measurement Units (IMU) measure the acceleration in six degrees of freedom. For the yaw motion, the angular $\hat{z}$ acceleration is integrated into an angular velocity and the velocity is integrated into angular displacement. Given a set initial heading, we can then assume that any angular displacement in yaw would mean a change in the heading of the agent. Micro-electro Mechanical Systems (MEMS) based IMUs have been thoroughly developed and tested and are onboard many manned and unmanned systems.

3.2 COTS Hardware and Open Source Software Design

This approach will include the open source ROS Indigo software architecture [17] for integration, COTS Clearpath Robotics Heron USV platform [18], the COTS Parrot AR.Drone UAV platform [19], and SwiftNav Piksi GPS [20] for RTK data. We begin with the discussion of software integration first because the choice of communication software drives the choices for hardware for this project.

3.2.1 ROS architecture

Robot Operating System (ROS) is an open source framework for robot operation development initially released in 2007 by Willow Garage. It functions as a publisher-subscriber architecture, where one or more machines may publish, or send out, certain information, and either the same or other machines may subscribe, or listen, to that information. It provides a structured communications between machines running different operating systems and hardware. It also has online and open source libraries and tools to allow developers to quickly utilize previous work in robot applications. ROS packages contain nodes which may receive, post and combine sensor, actuator, state and other information. This communications path can be accomplished wirelessly, as long as the agents are on the same wireless network.

Figure 3.1: Parrot ARDrone Version 2.0 Source: [19].

Parrot ARDrone Specifications	
Dimensions	451 x 451 mm
Processor	1GHz 32 bit ARM Cortex A8
Total Weight	380 g
Max Speed	21.6 kts (11.11 m/s)
Payload	~200 g
Battery	1500 mA/H LiPo rechargeable
Rotor Power Draw	14.5 W
Nominal Use Runtime	15 min

Table 3.1: Quadrotor Drone Specifications.

3.2.2 Quadrotor Drone

The ARDrone 2.0 (Figure 3.1) will be used as the agent experiencing the artificial spring force. The ARDrone is a quadrotor aircraft from the French company Parrot. It was originally launched in 2010, with version 2.0 launching in 2012.

Through control of the four propellers, the drone can move with six degrees of freedom: forward/back, vertical, lateral, roll, pitch and yaw. Battery capacity and weight affect the flight time of the ARDrone. In addition to the ARDrone's 380 g mass, there is also the protective shell, the battery and the user-added external RTK GPS module and antenna which brings the total mass of the roving agent to 980 g. For the purposes of this experiment, the average flight time will be assumed as 15 minutes with a mass of 1 kg (Table 3.1).

The drone also comes with two integrated cameras, one forward facing and one downward facing (neither of which will be used for this study).

Internal to the drone are numerous sensors of interest. Pressure sensors measure the ambient barometric pressure giving an estimation of altitude, a magnetometer that, when calibrated, gives the magnetic heading of the drone, and MEMs based IMUs give accelerations in 6 degrees of freedom. The angular acceleration is integrated to give angular velocity measurements and again integrated to give pitch, roll and yaw angle. Take-off, landing, hovering and trimmed flight are all accomplished through the pre-programmed onboard processes, the only requirement is the command to do so. Control of the drone is accomplished remotely through the onboard 802.11n ad hoc Wi-Fi network. Normally, one would use an iPad or Android device, download the respective applications, connect to the ARDrone and begin the flight.

There are many quadrotor aircraft on the market today, but what makes the ARDrone interesting for this research is the third-party uses. Parrot publicly released the ARDrone Software Development Kits (SDK), which allows third-party developers to write their own applications for controlling the drone. This will be discussed in the next chapter on system integration.

3.2.3 USV

The Heron (Figure 3.2) will be used as the USV. It will act as the base station from which the agents will determine their relative distance from each other.

Figure 3.2: Clearpath Robotics Heron USV Source: [18].

Clearpath Robotics Heron Specifications	
Dimensions	1.35 x 0.98 x 0.32 m
Total Weight	30 kg
Max Speed	3.3 kts (1.7 m/s)
Payload	10 kg
Battery	29 a/H NiMH rechargeable
Thruster Power Draw	70 W
Nominal Use Runtime	2.5 H

Table 3.2: Surface Vessel Specifications.

Heron was launched in 2014 from Clearpath Robotics. It is a catamaran design with differential thrusters for propulsion and steering. It comes ROS-ready, where onboard sensors and actuators are available as ROS topics. Heron can support up to 10 kg of payload (Table 3.2), more than enough for an ARDrone landing area and base station equipment. Sensors onboard include GPS, LiDAR, IMUs, front facing and PTZ (Pan-Tilt-Zoom) camera, temperature, pressure and more. For the purposes of this research, we will only be using the onboard GPS sensor to drive a predetermined waypoint course. Control of the USV is ac-complished through the onboard 802.11 Wi-Fi network, with a long range wireless network station that provides Wi-Fi at distances up to 1km.

3.2.4 GPS-RTK Module

The GPS-RTK module that will be used is the Piksi GPS receiver from SwiftNav (Figure 3.3). The small, light weight, low-power consumption and fast position solution update rate make it ideal for use on agents and for this research. Used alone, the Piksi module would function as any other GPS receiver, giving standard fix information. However, when used in conjunction with another module, the modules will share GPS carrier phase information via the 915 MHz radio link. This data is then used to provide relative position from one module to the other with centimeter level accuracy. The UHF radio link can provide over 1 km of range between the rover agent and the base station.

SwiftNav provides the software for visualizing the RTK fix positioning between the two radio connected modules. In the next chapter, we will discuss how the position information

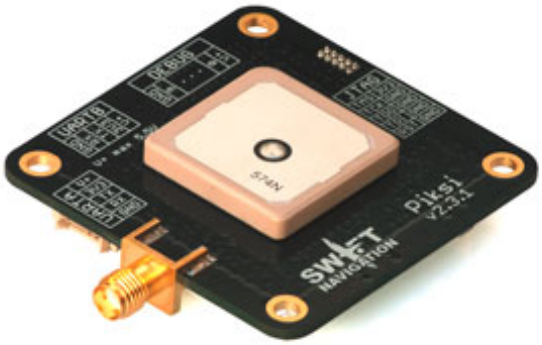

Figure 3.3: SwiftNav Piksi Module Source: [20].

SwiftNav Piksi Specifications	
Dimensions	53 x 53 mm
Processor	168 MHz STM 32F4 ARM Cortex-M4 DSP
Total Weight	32 g (226 g with GPS high gain antenna and UHF Telemetry Radio)
Power Draw	500 mW
Supply Voltage	3.3-5.5 V
Accuracy	1-5 cm (relative)
Solution Update Rate	50 Hz

Table 3.3: RTK GPS Module Specifications.

is sent to the artificial physics algorithm.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
System Integration

With the RTK GPS modules and onboard IMUs, we have position and orientation information of the agent relative to the base station. We must get the position and orientation information of the current state of the agent to the spring-force algorithm to determine the required velocity vector for each agent. Once the velocity vector is determined, we must then send the respective velocity command to each agent to accomplish the desired behavior.

4.1 Agent Velocity Control and Sensor Data

To establish a communications path of real-time information and required commands, a ROS node must be set up on each agent. Mani Monajjemi from Simon Fraser University, released a ROS driver for the Parrot ARDrone [21]. This driver sets up publisher-subscriber node, which publishes onboard sensor information (IMU information, battery life, etc.) and subscribes to (or listens for) velocity commands sent to the drone. The open source software `ardrone_autonomy`[1] package allows COTS ARDrones to be set up as a ROS node. The ardrone autonomy package is designed for a single instance of the ardrone node. Each ARDrone acts as its own wireless router and provides its own ad hoc wireless network. One would connect to the ARDrone network via a computer, load the ardrone ROS driver, and control (via ROS) one agent from one computer.

4.2 Multiple Agent Network

If multiple agents are to be controlled from a single station, more work must be done. NPS research assistant Mike Clement [22] put forth an approach to accomplish this. Each of the ARDrones are set up via telnet to shut down its local wireless network and connect to a separate wireless access point and given a specified IP address. By designating unique IP addresses, numerous ARDrones to be connected to the same wireless network. A Python script scans the wireless network looking for the pre-set IP addresses for connected agents

[1] https://github.com/AutonomyLab/ardrone_autonomy

and, once a agent is found, load one instance of the ardrone autonomy ROS driver for each agent. Unique node and topic names for the agents must also be identified in ROS, so that each velocity command generated in the spring algorithm can be sent to the proper agent. This allows the system to be able to control (via ROS) multiple agents near simultaneously.

However, out of the box ARDrones are set up to use the same User Datagram Protocol (UDP) network ports. So, if two or more agents are used, they could be trying to send their information on the same port, causing interference and dropped packets. Mike Clement [22] also set up a UDP remapping tool to resolve this problem by specifying the port address that each ARdrone is assigned. Now, practically speaking, we can get the heading information of the agent(s) into the artificial physics algorithm and we can send velocity commands to the agents as well.

4.3 Relative Position Updates

Still lacking is the position information. The SwiftNav software, when a Piksi module is connected, allows the user to see information about the GPS signals received by the modules. Once the RTK GPS fix solution has been obtained via the embedded software, user can see the North-South (y) and East-West (x) location of one module relative to the other. One module is placed on the agent as a rover and the other module is designated as a base station. But just being able to see the x-y location of the agent does us no good; we need to be able to feed that information in near real-time into the artificial physics algorithm. A ROS driver, written by Paul Bouchier [23], publishes the x and y coordinates of the RTK fix as a ROS topic and is available open source on GitHub as the `swiftnav_piksi`[2] driver package. Again, a unique identifier is required for each RTK node and topic for position update information. By changing the group namespace in the launch file of the package, we ensure that the RTK information is uniquely identified for each agent.

4.4 MATLAB and ROS

The Robotics Systems Toolbox is an add-on kit for MATLAB that starts a ROS node and can publish and subscribe to topics from a MATLAB script. After initializing the node, the spring force algorithm subscribes to the RTK position information, applies the virtual

[2]https://github.com/PaulBouchier/swiftnav_piksi

spring force, and publishes the required velocity command to the motor controller of each agent.

4.5 Experimental Setup

This system is set up so that the base station, which consists of a computer and a Piksi module, is independent and mobile (Figure 4.1).

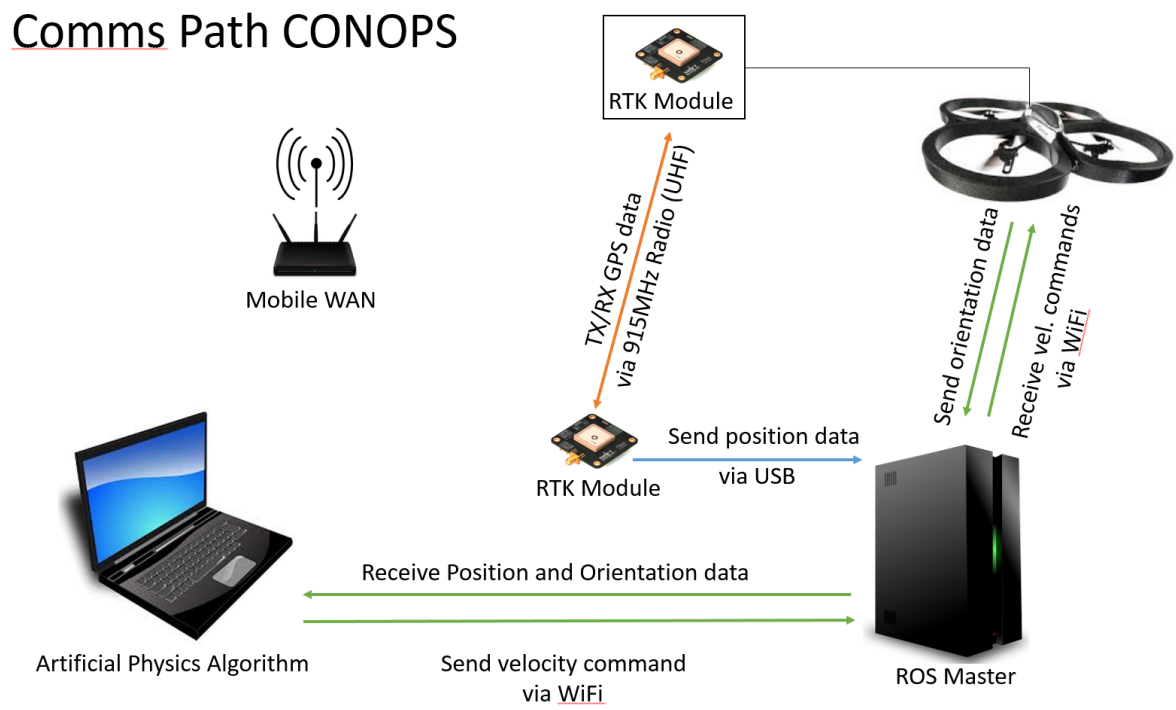

Figure 4.1: Concept of Operations for Communication Path.

For the scenario put forth in this paper, the base station is put on Clearpath Robotics Heron USV, to allow take off, following and landing multiple agents on a USV. The Heron is controlled through a separate ROS node. A simple box perimeter following routine is setup for the Heron to accomplish while the agents accomplish the swarm behavior at the same time.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Experimental Results

With a tested spring force algorithm computer model, functioning hardware, and a reliable mobile network, we are ready to test the behavior of real-world agents in an outdoor environment. For the first three tests, a single agent is made to move in one dimension at a location relative to the base station. The following "bird's eye view" diagrams are a MATLAB graphical representation (Figure 5.1) of relative longitude in meters (x-axis) and relative latitude in meters (y-axis) as referenced from the base station located at the origin. For example, a value of x = 5 would correlate to 5 meters East of the base station, while a value of x = -5 would correlate to 5 meters West of the base station. The rover (designated as the green dot) moves along the +y axis (i.e., North-South axis) only. The base station (designated as the red dot) is set at the origin.

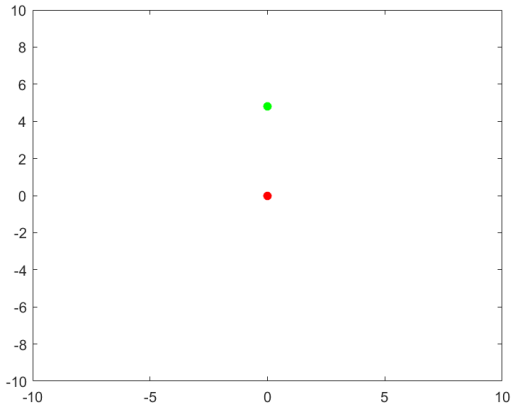

Figure 5.1: Roving Agent (Green) and Base Station (Red).

The position information of the roving agent is sent to the spring force algorithm, and the output is the velocity command required for the agent to maintain the desired separation distance. For all the following tests, the sping constant k is set to 2 N/m and the damping coefficient β is set to 2.8 kg/s, making the system critically damped.

5.1 Test 1: Validation of Spring Force Algorithm with RTK Input (θ = 0, Desired Separation = 5m)

For the first test (Figure 5.2), the agent is located North and faces North, with the y-axis of the plot is designated as the North-South axis. Any output velocity command from the algorithm should be a +/- y velocity (i.e., the agent is located north of the base station and is facing north; any movement should be simply forward or backward). Note: The agent is not moving freely, the author is walking the agent North and South to check the response of the system.

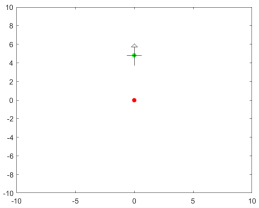

Figure 5.2: Position of Agent and Base Station ($\theta = 0$).

At the farthest distance away (about 9 meters), the artificial physics algorithm calculates a force of 8 N facing toward the base station, pulling it in closer (Figure 5.3).

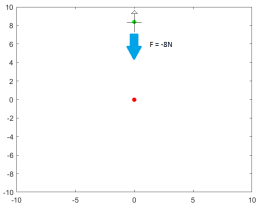

Figure 5.3: Resulting Artificial Force on the Agent ($\theta = 0$) at Farthest Distance.

At its closest distance to the base station (about 3 meters), the artificial physics algorithm calculates a force of 4 N facing away from the base station, pushing it farther away (Figure 5.4).

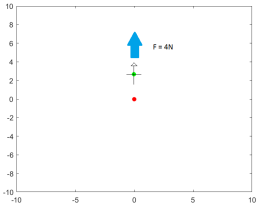

Figure 5.4: Resulting Artificial Force on the Agent ($\theta = 0$) at Closest Distance.

The resultant force is integrated using techniques described in Chapter 2. The output is a velocity command, which is then sent to the agent (Figure 5.5).

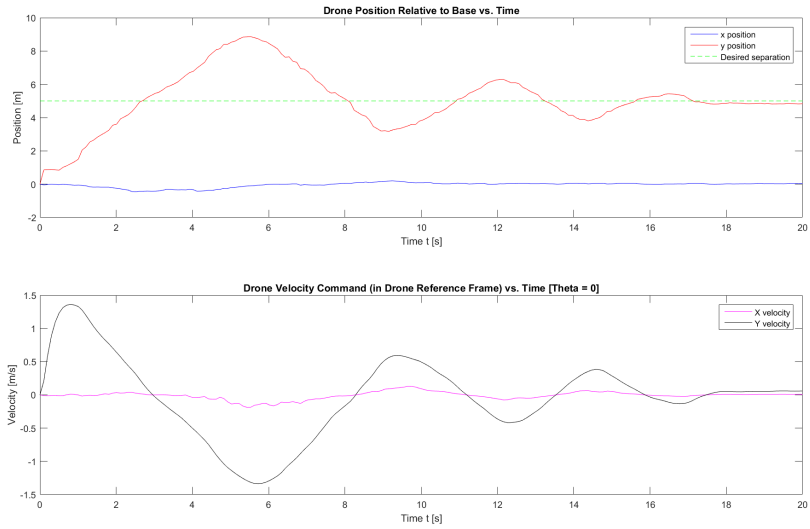

Figure 5.5: Position of Agent and Resulting Velocity Command ($\theta = 0$).

We can see that if the agent position is closer than desired separation of 5 meters, the y velocity command is positive (i.e., if the agent is too close, then move forward). If the agent is farther than the desired separation, then the y velocity command is negative (i.e., if the agent is too far away, then move backward). There is minor movement in the x axis, but that is a result of ambient environmental factors and should be disregarded.

5.2 Test 2: Validation of Spring Force Algorithm with Rotational Input (θ = 90, Desired Separation = 5m)

For the next test (Figure 5.6), the agent was again made to move on the +y axis relative to the base station (i.e., North), but this time the agent would be facing 90 degrees counter-clockwise (i.e., due East). Again, the agent is not moving freely, the author initially rotated the agent and then walks the agent North and South to check the response of the system.

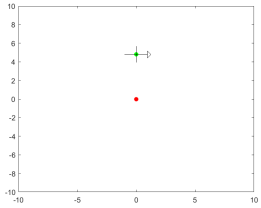

Figure 5.6: Position of Agent and Base Station ($\theta = 90$).

As before, at the farthest distance away, the artificial physics algorithm calculates the same force of 8 N facing toward the base station (Figure 5.7).

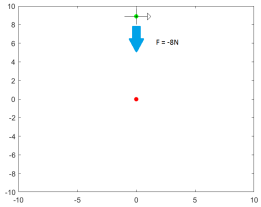

Figure 5.7: Resulting Artificial Force on the Agent ($\theta = 90$) at Farthest Distance.

Results are similar as the previous experiment for the closest approach of the agent to the base station (Figure 5.8).

Since the agent has rotated 90 degrees, the resulting velocity command (Figure 5.9) should now be given (as a result of the rotation transformation matrix) as a +/- x velocity (i.e., any movement should be right or left, since the reference frame of the agent has changed).

We can see that if the agent position is closer than desired separation of 5 meters, the x velocity command is negative (i.e., if the agent is too close, then move left). If the agent

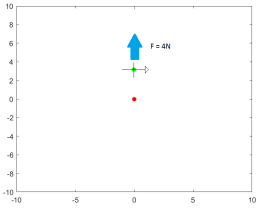

Figure 5.8: Resulting Artificial Force on the Agent ($\theta = 90$) at Closest Distance.

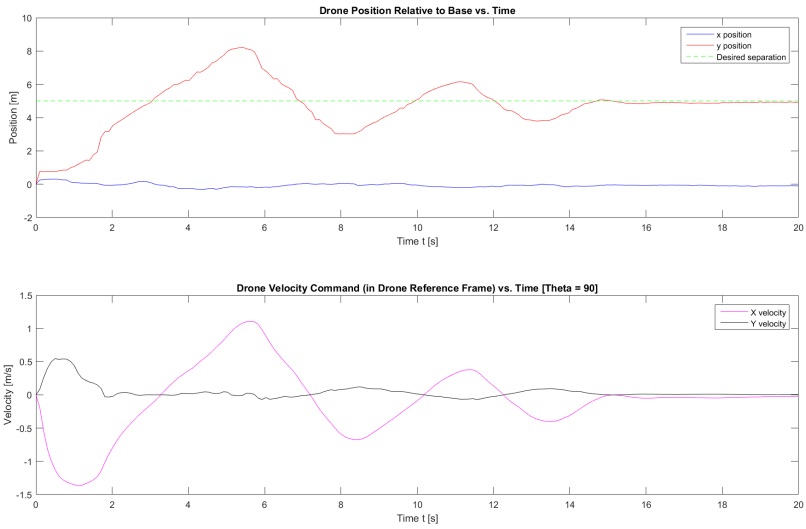

Figure 5.9: Position of Agent and Resulting Velocity Command ($\theta = 90$).

is farther than the desired separation, then the x velocity command is positive (i.e., if the agent is too far away, then move right). There is minor movement in the x axis during the first few seconds because of the agent turning from facing North to facing East to start the test.

5.3 Test 3: Validation of Computer Model from real-world Dynamics

For the next test, we want to validate that our computer model accurately reflects the real-world dynamics of the quadrotor aircraft. To do that, we input a step velocity command in the vertical direction to the computer model and to the actual agent (Figure 5.10). The +y axis of the 2D plot is now designated as the vertical height above ground, and the agent moves only up and down. The ultrasonic altimeter on the agent gives the height above ground position of the agent in mm.

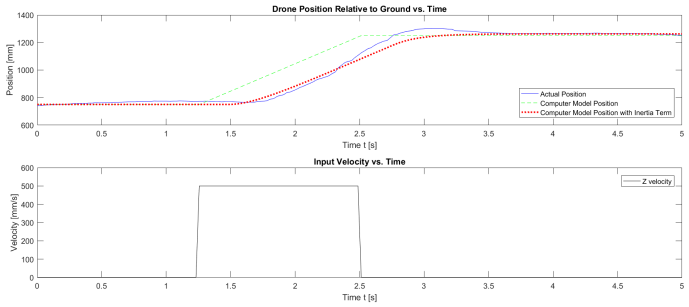

Figure 5.10: System Response Velocity Step Function.

Because the real-world system must deal with physical inertia of spinning up the rotors and also the time latency of the ROS command, an obvious delay is seen between the computer model and the real-world response. To compensate, we add an "inertia" term, a first order transfer function in the Laplace domain with a time constant $\tau = 0.2$ (Figure 5.11).

The original computer model had as high as a 25% error between the predicted value and the actual value. With the inertia term added, we can see that the error went only to around 5%. With the inertia term, we can have good confidence that the computer model will closely resemble the real-world dynamics of the system.

5.4 Test 4: Validation of Critically Damped System

Now, we allow free movement of the agent to test our previous prediction of a critically damped system where $k = 2$ N/m and $\beta = 2.8$ kg/s. We put the spring in the vertical direction with the reference point or base station as the ground. The +y axis of the 2D plot remains designated as the vertical height above ground, and again the agent only moves up

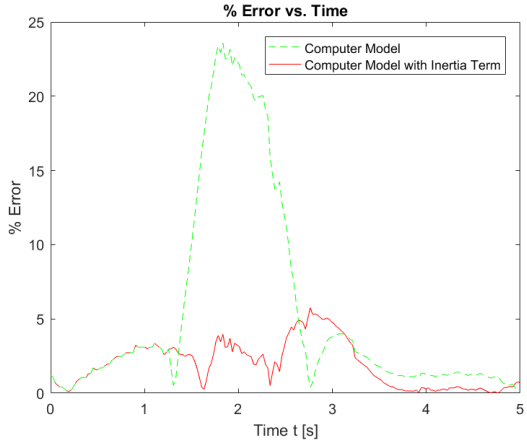

Figure 5.11: Error between Computer Model and real-world Response.

and down. Desired separation is set to a height of 1.5 m after an initial hovering altitude of 0.75 m. The computer model predicts the behavior of the system (Figure 5.12).

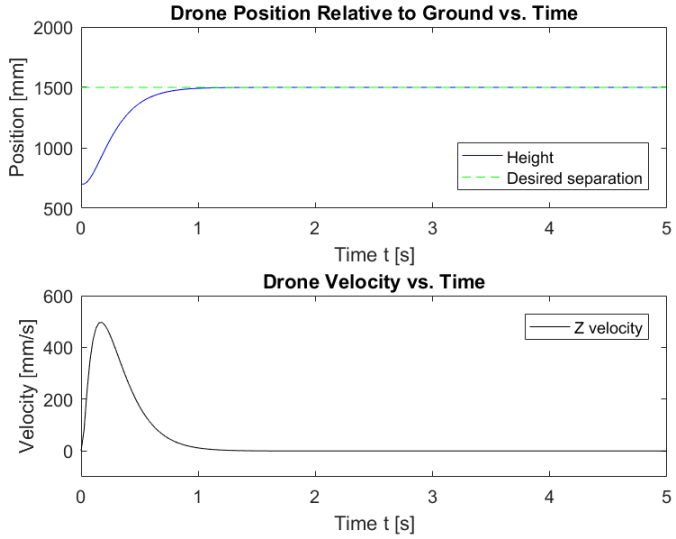

Figure 5.12: Simulation of Vertical Spring (Critically Damped).

The simulated agent rise time to the desired distance of 1.5 m was about 0.75 seconds.

For the real-world test, the ultrasonic altimeter sensor onboard the agent sends its altitude position information to the spring algorithm, and the resultant velocity is sent as a command

to the agent (Figure 5.13).

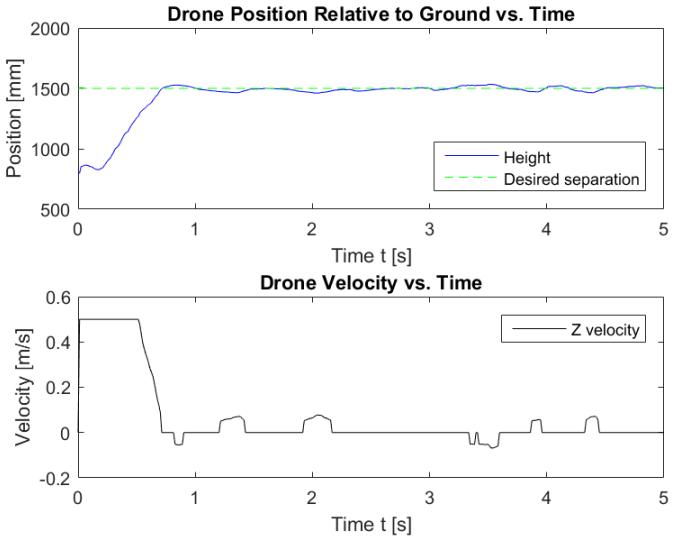

Figure 5.13: Actual System Response to Vertical Spring (Critically Damped).

The actual agent rise time to the desired distance of 1.5 m was also about 0.75 seconds. The upper limit of the velocity command sent to the agent is set to 0.5 m/s so as to not burn out motors or crash into a overhead light. We can also see very little overshoot due to the critical damping of the system. Some minor perturbations to the system can be seen after reaching steady state due to environmental effects, but similar perturbations are very common in real-world operations.

The agreement between the computer model and the real-world behavior gives credibility to the system architecture and any future tests.

5.5 Platform Substitution

Since the GPS information from a single RTK module must be sent over a UHF radio frequency, the amount of data traffic on that frequency increases when trying to scale up the number of units. This causes major processing problems with the system, since it is

trying to resolve since unique GPS data from separate units. The solution is to change the computer processing the RTK information from the base station to the agent. This allows the base station to broadcast its own location, and then the agents determine their relative location from that signal. However, even a small microcomputer to accomplish the RTK processing, in addition to the Piksi module and high gain antenna, proved to be too heavy for flying ARDrone. We made the decision to substitute the ARDrone quadrotor aircraft with a four-wheeled ground robot called Pioneer 3-AT (Figure 5.14).

Figure 5.14: Pioneer 3-AT.

The Pioneer is controlled by the same ROS publisher message, so the same script used for the previous experiments with the ARDrone can be used for the Pioneer. However, the quadcopter accepts velocity commands in six degrees of freedom, as mentioned before. The ground robot only accepts velocity commands in two degrees of freedom: forward-backward motion and angular $\hat{z}$ (yaw) motion. This substitution gives many more opportunities to test, while still validating the intent of close cooperative swarm engagements.

5.6 Test 5: Multi-agent Robotic Swarm Following

For the final test of this study, we perform a full demonstration of controlling multiple robots using an artificial spring with RTK GPS as the positional input information.

We start with three pioneer robots (conveniently named Robot1, Robot2 and Robot3) about 5 meters North of the base station, facing North (Figure 5.15). The "bird's eye view" diagram is still valid, with each of the robots' position located relative to the base station at the origin.

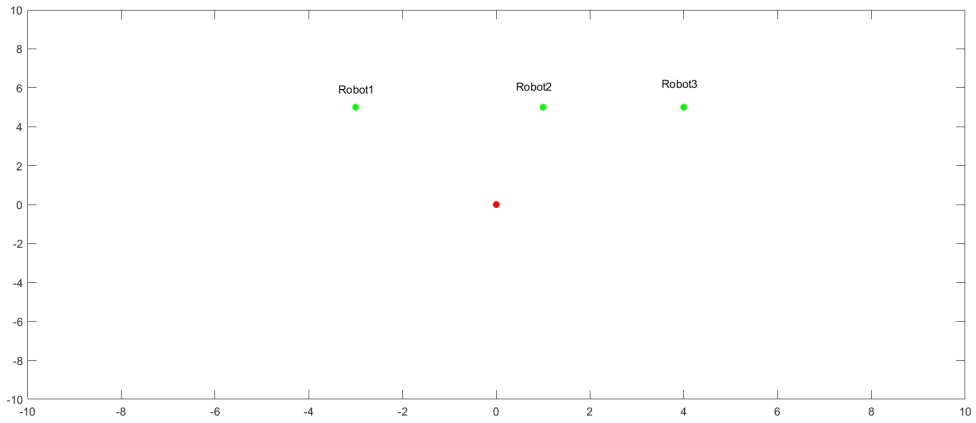

Figure 5.15: Multi-agent Test at Time=0 Seconds.

The $\hat{y}$ direction of the artificial spring is set to zero. This causes the robots to feel an attractive force in the y direction toward the base station at all times (Figure 5.16).

The $\hat{x}$ direction of the spring is ignored because of the ground robot does not have the six degrees of freedom as stated before. Although one does not have to ignore the $\hat{x}$ direction, but to accomplish a movement in East-West direction the robot would have to pivot to face that direction and then move. For simplicity's sake and proof of concept purposes, we simply do not send the $\hat{x}$ command to each respective robot.

The robots move toward the stationary base station until the difference in the y-position is (close to) zero (Figure 5.17).

The base station then moves North and South at a slow walking pace. As the base station

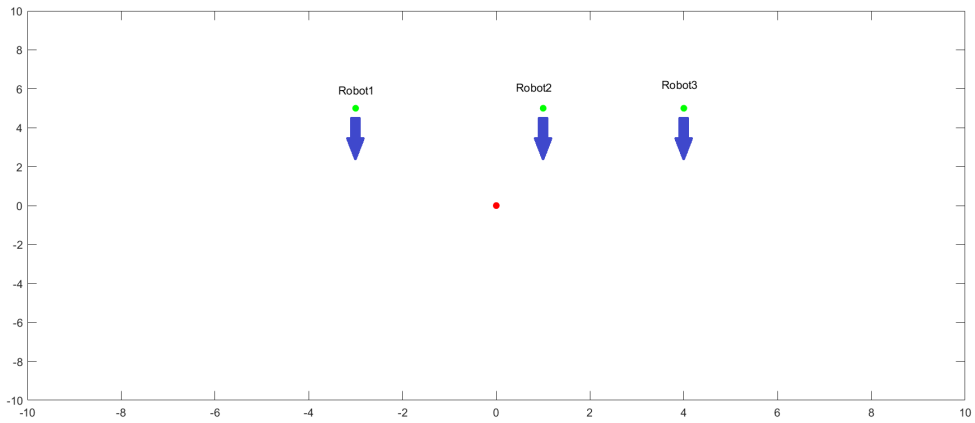

Figure 5.16: Multi-agent at Time=1 Second.

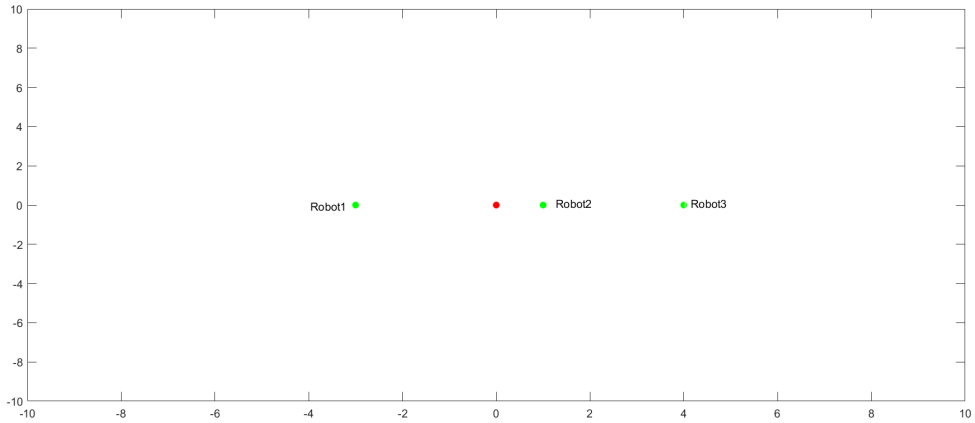

Figure 5.17: Multi-agent Test at Time=8 Seconds.

moves North (in the $+y$ direction), the robots also move toward the North in order to maintain as close to a zero difference in the y-position as possible (Figure 5.18)

The same can be seen as the base station moves South (in the $-y$ direction) (Figure 5.18).

The full raw data plot of position information and velocity commands for each of the respective robots can be seen in Figure 5.19.

This proves that a multi-agent robotic swarm system can be controlled using an artificial

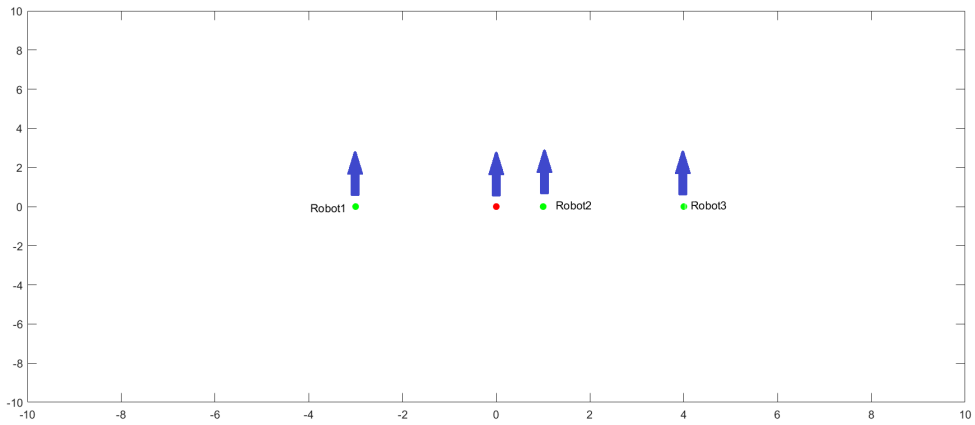

Figure 5.18: Multi-agent Test at Time=12 Seconds.

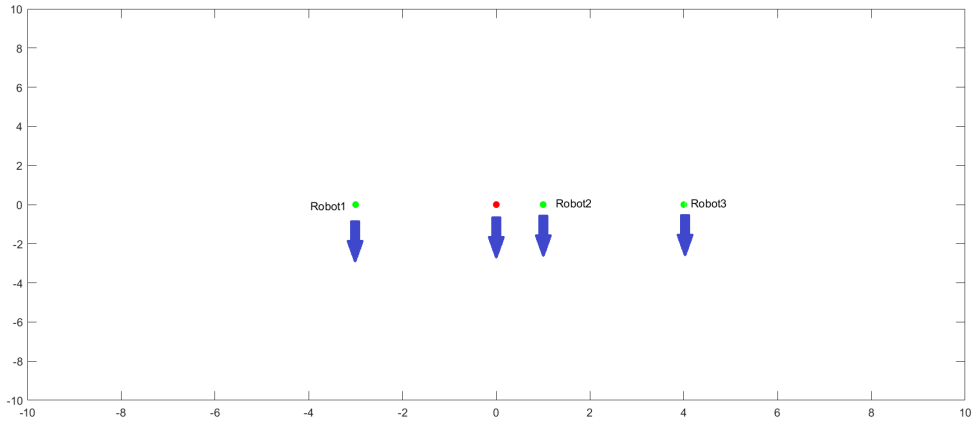

Figure 5.19: Multi-agent Test at Time=12 Seconds.

spring algorithm with RTK GPS position inputs.

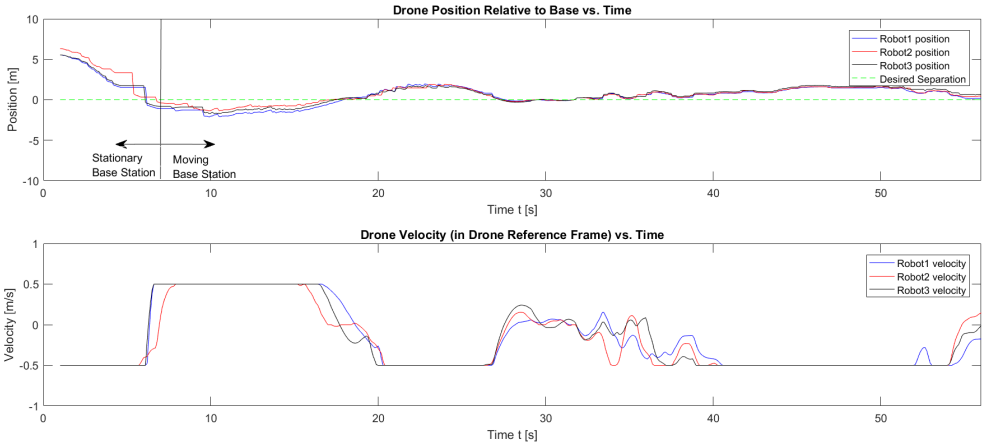

Figure 5.20: Position and Velocity vs. Time (Three Agents, Spring Length = 0).

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Conclusion

6.1 Sensor Issues

Although the RTK GPS information is very useful for precise robotic control, the two modules need to be receiving signals from numerous GPS satellites. In practice, this means the modules need a clear view of the sky. If the robot goes under a tree, or too close to a tall building which obstructs a large portion of the sky, the RTK GPS fix information be-comes unreliable. Sensors such as GPS, LIDAR, IMUs and others must be evaluated for their strengths and weaknesses and then combined in order to provide a system with a high degree of accuracy and reliability.

The IMUs on the ARDrone did not provide the necessary reliability for a robust system. The angle of orientation from the onboard IMU jumps around drastically during takeoff, most likely due to the spike in electromagnetic interference from power sent to the brushless motors. In addition, the angle would precess on the order of 10 degrees per minute, but that too would vary. Some compensation was added to the script, but determining orientation would be much easier with a better IMU.

Yet another issue with the IMU is the fact that the ARDrone must be pointing East upon start up in order for the x-axis of the RTK GPS fix and the x-axis of the agent to line up. Otherwise, the algorithm will, for example, send a North command to the agent, but the agent will interpret it as a command to fly to the West.

The obvious fix would be to use a compass to determine orientation. The ARDrone does have a magnetometer onboard, which would solve the IMU problem by providing an absolute heading. However, when publishing on a ROS topic, the magnetometer gets stuck at a certain heading and becomes unreliable. A better magnetometer, or a different platform, would go a long way in alleviating these headaches in the future.

Because of the issues with the ARDrone sensors as well as the stability problems because

of the additional weight, we did not have the confidence to attempt waterborne formation maneuvers. However, the approach outlined can be applied to waterborne operations, with better platforms and sensors.

6.2 Future Work

A quadrotor platform with a higher payload capacity and more reliable IMUs/magnetometers could in theory accomplish the stated goal of having multiple UAVs launch from a USV and executing a task using the method put forth in this research. In addition, the launch and recovery of fixed-wing autonomous agents from a USV would offer much greater range, flight time and payload capacity than a quadrotor aircraft. The techniques described in this paper can also easily be applied to fixed-wing aircraft, with the obvious understanding that they do not have the ability to hover.

With the IMU data, the RTK GPS data, and our predictive algorithms, a natural progression would be the introduction of a Kalman filter. This would give better confidence of the true position of the agent even if sensor data from one source is degraded, as the RTK GPS data would be if traveling under tree cover. Additional sensors such as LIDAR and SONAR can also be included to detect obstacles that are obviously not broadcasting their GPS location on our ROS network.

The Navy can greatly benefit from the use of unmanned systems accomplishing increasing roles and responsibilities in mission related tasks. Algorithms that introduce a high level of autonomy relieve the burden of direct control of a robot, allowing for the use of numerous agents and allowing the operator to focus on big picture strategic issues.

APPENDIX: MATLAB CODE

MATLAB script for Artificial Spring with three agents using ROS toolkit (with RTK position publishers and velocity command subscribers).

```
rosinit

function [ ForceX, ForceY ] = hForce2D( x1, y1, x2, y2, r0, k )
distR = sqrt((x1-x2)^2 + (y1-y2)^2);
forceTemp = -k*(distR-r0);
ForceX = forceTemp*(x1-x2)/distR;
ForceY = forceTemp*(y1-y2)/distR;
end

function rtk1(~, message)
   % Declare global variables to store position and orientation
   global x1
   global y1
   global z1

   % Extract position and orientation from the ROS message and assign the
   % data to the global variables.
   x1 = [message.Pose.Pose.Position.X];
   y1 = [message.Pose.Pose.Position.Y];
   z1 = [message.Pose.Pose.Position.Z];
end

function rtk2(~, message)
   global x2
   global y2
   global z2
   x2 = [message.Pose.Pose.Position.X];
   y2 = [message.Pose.Pose.Position.Y];
   z2 = [message.Pose.Pose.Position.Z];
end
```

```
function rtk3(~, message)
    global x3
    global y3
    global z3
    x3 = [message.Pose.Pose.Position.X];
    y3 = [message.Pose.Pose.Position.Y];
    z3 = [message.Pose.Pose.Position.Z];
end

swarm_size = 3;
mass = 1; %mass of one drone
t_s = 0.1; %time step size in seconds
iter = 10000; %number of time steps
beta = 2.8; %damping constant
k = 2; %spring constant of agents
k1 = -2; %spring constant of base station
r0 = 7; %separation from other agents
r1 = 1; %separation from base station

 % initialize starting velocity and acceleration
[dampForceX,dampForceY,X_vel,X_vel_xfrm,X_accel,Y_vel,Y_vel_xfrm,Y_accel,
x,y,theta] = deal(zeros(iter,swarm_size));

 % Create a subscriber for the RTK fix topic
rtkfix1 = rossubscriber('/gps/rtkfix', @rtk1);
rtkfix2 = rossubscriber('/gps_2/rtkfix', @rtk2);
rtkfix3 = rossubscriber('/gps_3/rtkfix', @rtk3);
global x1 x2 x3 y1 y2 y3

 % Create a publisher for the ROS TWIST message
publisher = rospublisher('/robot1/cmd_vel', 'geometry_msgs/Twist');
twist = rosmessage(rostype.geometry_msgs_Twist);

publisher2 = rospublisher('/robot2/cmd_vel', 'geometry_msgs/Twist');
```

```
twist2 = rosmessage(rostype.geometry_msgs_Twist);

publisher3 = rospublisher('/robot3/cmd_vel', 'geometry_msgs/Twist');
twist3 = rosmessage(rostype.geometry_msgs_Twist);

[twist.Linear.X,twist.Angular.Z,twist2.Linear.X,twist2.Angular.Z,
twist3.Linear.X, twist3.Angular.Z] = deal(0);

send(publisher,twist);
send(publisher2,twist2);
send(publisher3,twist3);

X_pos_OP = 0; %base station at origin
Y_pos_OP = 0;

pause(1)

 % % Main Loop
for j=2:iter
  for i=1:swarm_size
       TotForceX=0;
       TotForceY=0;

       % get rtk info in X-Y (East - North) coordinates
       x(j,1) = x1;
       y(j,1) = y1;

       x(j,2) = x2;
       y(j,2) = y2;

       x(j,3) = x3;
       y(j,3) = y3;

    for index=1:swarm_size
     if index~=i
```

```
  [ForceX, ForceY] = hForce2D(x(j,i),y(j,i), x(j,index), y(j,index),
   r0, k);
  [OP_ForceX, OP_ForceY] = hForce2D(X_pos_OP, Y_pos_OP,
  x(j,index), y(j,index), r1, k1);
  TotForceX= TotForceX + ForceX + OP_ForceX;
  TotForceY= TotForceY + ForceY + OP_ForceY;
 end
end

dampForceX(j,i) = TotForceX - beta.*X_vel(j-1,i);
dampForceY(j,i) = TotForceY - beta.*Y_vel(j-1,i);

X_accel(j,i) = dampForceX(j,i)/mass;
Y_accel(j,i) = dampForceY(j,i)/mass;

X_vel(j,i) = X_vel(j-1,i) + 0.5.*(X_accel(j-1,i)+X_accel(j,i)).*t_s;
Y_vel(j,i) = Y_vel(j-1,i) + 0.5.*(Y_accel(j-1,i)+Y_accel(j,i)).*t_s;

%reference frame transformation
X_vel_xfrm(j,i) = (X_vel(j,i).*cosd(theta(j,i))
                        + Y_vel(j,i).*sind(theta(j,i)));
Y_vel_xfrm(j,i) = (-X_vel(j,i).*sind(theta(j,i))
                        + Y_vel(j,i).*cosd(theta(j,i)));

twist.Linear.X= X_vel_xfrm(j,1);
twist2.Linear.X=X_vel_xfrm(j,2);
twist3.Linear.X=X_vel_xfrm(j,3);
twist.Linear.Y= Y_vel_xfrm(j,1);
twist2.Linear.Y=Y_vel_xfrm(j,2);
twist3.Linear.Y=Y_vel_xfrm(j,3);

send(publisher,twist);
send(publisher2,twist2);
send(publisher3,twist3);
```

```
  clf;
  h=plot(x(j,1),y(j,1), 'o');
  set(h,'MarkerEdgeColor','none','MarkerFaceColor','g')
  hold on
  h2=plot(x(j,2),y(j,2), 'o');
  set(h2,'MarkerEdgeColor','none','MarkerFaceColor','g')
  h3=plot(x(j,3),y(j,3), 'o');
  set(h3,'MarkerEdgeColor','none','MarkerFaceColor','g')
  h1=plot(X_pos_OP, Y_pos_OP, 'o');
  set(h1, 'MarkerEdgeColor','none','MarkerFaceColor','r')
  axis([-10 10 -10 10]);

  pause(.1);

end;
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] Economist. (2011, Oct.). Flight of the drones: Why the future of air power belongs to unmanned systems. [Online]. Available: http://www.economist.com/node/21531433/. Accessed April, 2016.

[2] DOD Open Publication Ref No 14-S-0553 (2013). Unmanned systems integrated roadmap fy2013-2038. [Online]. Available: www.acq.osd.mil/sts/docs/DoD%20USRM%202013.pdf/. Accessed April, 2016.

[3] D. Smalley. (2014). Navy's autonomous swarmboats can overwhelm adversaries. [Online]. Available: http://www.onr.navy.mil/Media-Center/Press-Releases/2014/autonomous-swarm-boat-unmanned-caracas.aspx/. Accessed April, 2016.

[4] S. Littlefield. Anti-submarine warfare continuous trail unmanned vessel (ACTUV). [Online]. Available: http://www.darpa.mil/program/anti-submarine-warfare-continuous-trail-unmanned-vessel/. Accessed April, 2016.

[5] Amazon.com Inc. (2014, May). Amazon Prime air. [Online]. Available: http://www.amazon.com/b?node=8037720011/. Accessed May 9, 2015.

[6] Alphabet Inc. (2016). Google self driving car project. [Online]. Available: https://www.google.com/selfdrivingcar/. Accessed April, 2016.

[7] Novatel. An introduction to gnss. [Online]. Available: http://www.novatel.com/an-introduction-to-gnss/chapter-5-resolving-errors/real-time-kinematic-rtk/. Accessed Jan, 2016.

[8] N. Hogan, "Impedance control: An approach to manipulation," in *IEEE International American Control Conference 1984*, 1984, pp. 303–1313.

[9] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. IEEE International Conference on Robotics and Automation*, 1991, pp. 1398–1404.

[10] W. Spears, D. Spears, J. Hamann, and R. Heil, "Distributed, physics-based control of swarms of vehicles," *Autonomous Robots*, vol. 17, pp. 137–162, 2004.

[11] T. Apker and M. Potter, "Robotic swarms as solids, liquids and gasses," in *AAAI Fall Symposium: Human Control of Bioinspired Swarms*, 2012.

[12] M. Ma'sum, G. Jati, M.K. Arrofi, A. Wibowo, W. Jatmiko, and P. Mursanto, "Autonomous quadrotor swarm robots for object localization and tracking," in *Proc. IEEE International Symposium on Micro0Nano Mechatronics and Human Science*, 2013, pp. 1–6.

[13] (2012). Vicon homepage. [Online]. Available: http://www.vicon.com/. Accessed May, 2016.

[14] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Toward a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, pp. 287–300, 2013.

[15] M.S. Alvissalim, B. Zaman, Z. Hafizh, M. Ma'sum, G. Jati, W. Jatmiko, and P. Mursanto, "Swarm quadrotor robots for telecommunication network coverage area expansion in disaster area," in *Proc. IEEE SICE Annual Conference 2012*, 2012, pp. 2256–2261.

[16] B. Campbell, "Human robotic swarm interaction using an artificial physics approach," M.S. thesis, Naval Postgraduate School, Monterey, California, 2014.

[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open source robot operating system," *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

[18] Clearpath Robotics Inc. (2014, Oct.). Heron unmanned surface vessel. [Online]. Available: http://www.clearpathrobotics.com/heron-bathymetry-unmanned-surface-vessel/. Accessed April, 2015.

[19] Parrot SA. (2012). Parrot ar.drone 2.0. [Online]. Available: http://www.parrot.com/usa/products/ardrone-2/. Accessed Feb, 2015.

[20] Swift Navigation Inc. (2016). Piksi real time kinematic gps oem. [Online]. Available: http://www.swiftnav.com/. Accessed Jan, 2016.

[21] M. Monajjemi. (2012). Autonomylab/ardroneautonomy. [Online]. Available: https://github.com/AutonomyLab/ardroneautonomy/. Accessed Sept, 2015.

[22] P. Bouchier. (2015, Nov.). Ros driver for swiftnav piksi. [Online]. Available: https://wiki.ros.org/swiftnav_piksi/. Accessed Jan, 2016.

[23] M. Clement. (2014, Feb.). Hacked 10 bits: Multiple ar. drones from a single computer using the ardrone autonomy ros package. [Online]. Available: http://hacked10bits.blogspot.com/2014/02/multiple-ardrones-from-single-computer/. Accessed Nov, 2015.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California